package com.twitter.home\_mixer.product.scored\_tweets.filter

import com.twitter.home\_mixer.model.HomeFeatures.InNetworkFeature

import com.twitter.home\_mixer.model.HomeFeatures.IsRetweetFeature

import com.twitter.home\_mixer.model.HomeFeatures.TweetUrlsFeature

import com.twitter.home\_mixer.product.scored\_tweets.param.ScoredTweetsParam.CompetitorURLSeqParam

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

import com.twitter.product\_mixer.core.functional\_component.filter.FilterResult

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.identifier.FilterIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.stitch.Stitch

object OutOfNetworkCompetitorURLFilter extends Filter[PipelineQuery, TweetCandidate] {

override val identifier: FilterIdentifier = FilterIdentifier("OutOfNetworkCompetitorURL")

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[FilterResult[TweetCandidate]] = {

val competitorUrls = query.params(CompetitorURLSeqParam).toSet

val (removed, kept) = candidates.partition(hasOutOfNetworkUrlFromCompetitor(\_, competitorUrls))

Stitch.value(FilterResult(kept = kept.map(\_.candidate), removed = removed.map(\_.candidate)))

}

def hasOutOfNetworkUrlFromCompetitor(

candidate: CandidateWithFeatures[TweetCandidate],

competitorUrls: Set[String]

): Boolean = {

!candidate.features.getOrElse(InNetworkFeature, true) &&

!candidate.features.getOrElse(IsRetweetFeature, false) &&

candidate.features

.getOrElse(TweetUrlsFeature, Seq.empty).toSet.intersect(competitorUrls).nonEmpty

}

}